AMENDMENTS TO THE SPECIFICATION:

Please substitute the following amended paragraph for the pending paragraph beginning on page 1, line 13:

The following U.S. patents are fully incorporated herein by reference: U.S. Pat. No. ~~5,871,731~~5,870,731 (Trif et al., "Adaptive Problem Solving Method and System"); U.S. Pat. No. 6,088,690 (Gounares et al., "Method and Apparatus for Adaptively Solving Sequential Problems in a Target System Utilizing Evolutionary Computation Techniques"); and U.S. Pat. No. 6,144,953 (Sorrells et al., "Time-Constrained Inference Strategy for Real-Time Expert Systems").

Please substitute the following amended paragraphs for the pending paragraphs beginning on page 19, line 24, through page 21, line 28:

Referring to Fig. 4, there is shown an embodiment of an adaptive constraint problem solver program. In this embodiment, problem P is provided to solving module ~~code~~ 420 and complexity module ~~code~~ 410 on path 460. The complexity-directed fine-grained, interleaved algorithm synthesis module ~~code~~ 430 of the problem solver resides within solving module ~~code~~ 420. Complexity module ~~code~~ 410 includes solver model 490, which is a data structure, for example a table, which contains configuration parameters C and expected behaviors B_e for different problems P. Path 450 provides solving module ~~code~~ 420 with configuration parameters C, in any known format, for example binary or ASCII. Configuration parameters C may be based on a utility function such as minimal solving time, optimal solution quality, etc. and represent changes in algorithm decision points or adjustment of algorithm parameters. Solver behavior may be time to solution, rate of objective improvement, number of search nodes, density of local minima, solution quality, or any other desired behavior type. Solving module ~~code~~ 420 makes choices in its algorithms that optimize the expected utility, and provide an actual behavior B_a on path 425 to comparison unit 445, which compares actual behavior B_a with expected behavior B_e, which complexity module ~~code~~ 410 provides along path 415. Path 440 provides complexity module ~~code~~ 410 with the difference in behavior ΔB determined by comparison unit 445, through comparison of data strings. For example, behavior may be represented as a single number (solving time so far) or an array of numbers (a history of the

values of the objective function, the function being optimized), or a more complicated data structure. The comparison may simply take the difference between the expected and actual solving time values, or compute the rate change of the array of objective function values and take the difference of that from an expected value or any other desired comparison. If necessary, complexity module–~~code~~ 410 then reconfigures the solving module–~~code~~ 420 with new parameters C provided along path 450, and solver model–~~code~~ 490 provides a new expected behavior B_e along path 415 to comparison unit 445. For the solution of a single problem P, the behaviors B_e, B_a, and ΔB and the configuration parameters C may be sent and read multiple times during the solving process. The final problem solution is sent to applications module along path 470.

Fig. 5 illustrates a flow chart, which describes the operation of the system illustrated in Fig. 4. In Fig. 5, problem P and initial configuration parameters C, received from solver model ~~code~~ 590, are sent to step 530, which sends the problem P and the parameters C to step 550 within solving module–~~code~~ 520. For the purpose of illustration, parameters C may specify, among other decision points, how many individual search steps the search method in solver iteration step 550 should execute. The complexity-directed fine-grained, interleaved algorithm synthesis module code of the problem solver resides within solver iteration step 550, but for clarity is not shown. Other types of parameters and heuristics C are possible, as will be appreciated by one skilled in the art. At step 550 the solver-~~code~~ attempts to solve the problem P. In this step, the solver code typically searches part of the search space, by using, for example, a gradient-descent, Nelder-Mead, interior-point, depth-first search, or any other technique that is intended to lead the solver towards promising regions of the search space, those regions where optimal valid solutions may be found. Step 550 depends on the type of solver or solvers used, as would be appreciated by one skilled in the art.

Once step 550 has been performed, step 560 checks the result to determine if a solution has been found, which is the case if no constraints are violated by the values of the solution and if no improvement in the objective function is found. If a valid solution has not been found, then the actual problem solver behavior is routed to step 570, where the performance so far is reviewed. Here, a decision is made as to whether another solver iteration step 550 should be executed, or whether the current status of the problem and the behavior B_a should be sent to the

complexity module ~~code~~ for an adaptation step. For example, the solving module ~~code~~ may have a fixed policy of running at least a minimum number of iterations before asking for adaptation. The actual behavior B_a and expected behavior B_e are compared at step 540 and resulting behavior ΔB is routed back to step 530. If the actual behavior B_a and the expected behavior B_e are different, parameters C are modified at step 530. For example, if the rate of improvement of the objective (specified in B_a) is lower than expected (specified in B_e), the number of individual search steps specified in C may be increased. The above process is then repeated until a solution is found.

Please substitute the following amended paragraphs for the pending paragraphs beginning on page 22, line 27 through page 24, line 2:

Referring now to Fig. 6, there is shown a flow chart describing the operation of an embodiment of the complexity-directed fine-grained, interleaved algorithm synthesis module ~~code~~. In Fig. 6, problem P and initial problem configuration parameters C, received from the complexity module ~~code~~, are sent to step 610 within the algorithm synthesis module ~~code~~ residing within the solving module ~~code~~. It should be appreciated that the core of the solver is the solver step 650. This solver step, which searches in the search space for the best point that might be a solution, generically makes use of a set of previously computed points in the search space. The calculation of these points, which are computed in steps 620 and 630, is in turn directed by a set of configuration parameter vectors, which are determined in step 610. Within step 610 a set of configuration parameter vectors is selected, discussed further hereinbelow.

For the first iteration, an initial set of default parameter vectors are selected and sent to step 620 for search initialization. Following search initialization, a partial search is performed at step 630 to compute a set of search space points before executing a solver step. This partial search step is meant to partially explore the search space and thus give a partial understanding of this space. For example, the partial search step may examine the immediate neighborhood of an initial (and possible solution) point in the search space and from that examination determine the gradient of the objective function, as may be done for constraint problems with continuous (real-valued) variables. The gradient is the difference in the function values of two points divided by their distance. As is known by those skilled in the art, by examining multiple points around a

single initial point, it is possible to determine in which direction the objective function changes the fastest and therefore a possible optimal solution might be located. As another example, the partial search step may examine the points in relation to the constraints and thus identify regions where the constraints are not violated. In yet another example, the partial search step may select a set of undetermined (open) variables of a partial solution, perhaps those variables with the smallest number of possible values or participating in the largest number of constraints, as may be done for constraint problems with discrete variables. Which kind of partial search step 630 should be performed is determined by the parameters which have been set in step 610.

Please substitute the following amended paragraph for the pending paragraph beginning on page 25, line 18 through page 26, line 25:

Referring now to Figs. 6 and 7, Fig. 7 is a flow chart of one embodiment of the steps taken within the configuration parameter vector code presented in Fig. 6 at step 610 to produce a fine-grained, interleaved synthesis of algorithms within the algorithm synthesis module code. Problem P and initial problem configuration parameters C, received from the complexity module ~~code~~, are sent to step 710 within the configuration parameter vector code residing within the algorithm synthesis module ~~code~~. At step 710 a determination is made as to whether this is an initial attempt to solve for configuration parameter vectors for the specified problem. If it is an initial attempt, default parameter vectors are selected at step 720, in the example of a local search, and the parameter vectors are sent to step 620 of Fig. 6 for search initialization and solution of the problem. A local search is performed with a loose termination criterion to approximately (and quickly) find a minimum of a simple model of a function near an initial point. On subsequent calls to step 610 in Fig. 6, alternative paths through the sample flow chart in Fig. 7 may be taken. At step 730 the behavior is evaluated. If the behavior is as predicted, then a decision is made at step 740 to retain the previous parameters, which are then sent to step 620 of Fig. 6. If the search fails to improve faster than expected (based on the complexity models), the problem solver identifies at step 750 whether only local searches have been performed. If only local searches have been performed, then at step 760 another point is selected at a distance far removed from the first point and another attempt is made to solve the problem. The local solver is used to determine a second nearby minimum. If this minimum is nearly the

same as the first, the problem solver concludes that the function is simple and that the local solver yields the best results. However, if the minima are not the same, then on a subsequent call at step 770 the minima may be used as the starting simplex for a Nelder-Mead algorithm. The parameters for the Nelder-Meand algorithm are set in step 780. At each point selected by the Nelder-Mead algorithm, a local search is made for the best nearby minimum. Nelder-Mead and the local solver handle global and local search, respectively. If the Nelder-Mead algorithm has trouble discovering the global structure, then on a subsequent call at step 790 parameters are set for a random search, for example, a search based on force-directed simulating annealing to generate better Nelder-Mead simplexes. Ultimately, if the function is very complex, the global search takes only stochastic steps with interleaved local searches. Such a hybrid, interleaved solver does much better than either algorithm separately. The transitions between local and global, between various local strategies, and between various global strategies can be made based on complexity and performance measures.

Please substitute the following amended paragraph for the pending paragraphs beginning on page 27, line 6:

Referring now to Figs. 6 and 8, Fig. 8 is a flow chart of an alternate embodiment of the steps taken within the configuration parameter vector code presented in Fig. 6 at step 610 to produce a fine-grained, interleaved synthesis of algorithms within the algorithm synthesis module ~~code~~. Problem P and initial problem configuration parameters C, received from the complexity module ~~code~~, are sent to step 810 within the configuration parameter vector code residing within the algorithm synthesis module ~~code~~. At step 810 a determination is made as to whether this is an initial attempt to solve for configuration parameter vectors for the specified problem. If it is an initial attempt, default parameter vectors are selected at step 820 for a local search and the parameter vectors are sent to step 620 of Fig. 6 for search initialization and solution of the problem. A local search is performed with a loose termination criterion to approximately (and quickly) find a minimum of a simple model of a function near an initial point. At step 830 the behavior is evaluated. If the behavior is as predicted, a decision is made at step 840 to retain the configuration parameters, which are then sent to step 620 of Fig. 6. If the search fails to improve faster than expected (based on the complexity models), then at step

850 revised parameter vectors based on actual behavior and the original problem statement and configuration parameters provided by the complexity module are selected and another attempt is made to solve the problem.

Referring now to Figs. 6 and 9, Fig. 9 is a flow chart of an alternate embodiment of the steps taken within the configuration parameter vector code presented in Fig. 6 at step 610 to produce a fine-grained, interleaved synthesis of algorithms within the algorithm synthesis module ~~code~~, with the added capability of learning. Problem P and initial problem configuration parameters C, received from the complexity module ~~code~~, are sent to step 910 within the configuration parameter vector code residing within the algorithm synthesis module ~~code~~. At step 910 a determination is made as to whether this is an initial attempt to solve for configuration parameter vectors for the specified problem. If it is an initial attempt, default parameter vectors are selected at step 920 for a local search and the parameter vectors are sent to step 620 of Fig. 6 for search initialization and solution of the problem. A local search is performed with a loose termination criterion to approximately (and quickly) find a minimum of a simple model of a function near an initial point. At step 930 the behavior is evaluated. If the behavior is as predicted, a decision is made at step 940 to retain the configuration parameters, which are then sent to step 620 of Fig. 6. If the search fails to improve faster than expected (based on the complexity models), then at learning step 950 the solver model residing within the complexity module ~~code~~ is updated based on actual behavior. For example, the difference may be used to refine the control parameters provided by the solver model ~~code~~. In one of many possible cases, if the rate of improvement of the objective is lower than expected, it may be recorded in the solver model that in the future all problems similar to the current problem should be solved with an increased number of individual search steps, or that a particular search method such as Nelder-Mead should be tried first instead of the default local search. This learning step depends on the type of parameter to be learned. Possible learning methods include value replacement, reinforcement learning, and any others known in the art. New parameter vectors based on the revised solver model configuration parameters provided by the complexity module are selected and another attempt is made to solve the problem.